***Full History Migration***

***TFS to Perforce***

# Introduction

This document describes the mechanical process of a full history migration from TFS to Perforce. Such migrations are usually the last stage of a larger process. The host of scheduling, representation trade-off, business factor mitigation, legacy application integration, and other factors that precede a successful migration are not addressed.

This document assumes that the reader has TFS and Perforce experience as both user and administrator. Most of the topics require a general knowledge of these systems. However some of the topics, such as representation considerations and recovery procedures, require an advanced level of knowledge.

# Overview

A full history migration from TFS to Perforce is accomplished using a set of tools that have been developed specifically for this purpose. Each tool within the set provides a specific functionality.

These are command line tools. If you are not familiar with using the command line it is likely that you’ll have challenges using these tools.

This section is about why and what. The remaining sections deal with how and when.

## Objectives

The tools were developed to meet these objectives:

File content equivalence.

File existence equivalence.

TFS changesets represented as Perforce changelists.

TFS check-in user, time, comments and notes available in Perforce submit.

Repeatable, Auditable, Recoverable.

Operation against live systems.

Front-door operation (perfmerge not required).

## Requirements

This section addresses the requirements for performing a full history migration.

### Infrastructure Software

These requirements reflect the earliest versions of infrastructure software that are known to support successful migrations.

TFS

* Version 2010 or later. 2012 strongly recommended
* All available patches are applied.
* Command line tool (tf)
* Power tools (tfpt) – potentially useful but not required.

Perforce

* Version 2012.1 or later
* Command line tool (p4)
* Perforce visual client (p4v) or command line visual tools (p4vc) – potentially useful for review of imported history.

Perl

* Version 5
* Release 5.16 or later

Perforce versions as early as 2010.1 and perl releases as early as 5.12 are likely to work. TFS versions prior to 2010 are unlikely to work.

### User Accounts

These user accounts are required to support an import:

A TFS user with unrestricted read access to all files in the collection(s) being migrated.

A Perforce super user with write access to the depot(s) being used for import. They should have a long (100 hour+) or unlimited timeout.

An optional Perforce user account with read access to the import depot(s) can be useful for examining intermediate and post-import results. Having read access prevents accidental interaction between this user and an import in progress.

### Disk space

These are the minimum disk space requirements for an import. If your process involves multiple imports – POC, test, stage, production, etc. – you need to adjust these requirements as appropriate:

A client system with sufficient disk space to hold three (3) copies of the most recent version of every file in the TFS collection.

The target Perforce server should have available depot disk space equivalent to the current size of the TFS repository.

The target Perforce server requires storage for metadata. A general estimate is 1K per unique TFS file being imported.

NOTE: The Perforce server space is specific to support of an import. Additional space is required for post-migration operation.

## Operational Context

The tools use the operational context of the command line environment in which they are operating. This simplifies establishing and validating operational context for experiment, debug, and import purposes.

### Collections

The tools are designed to work on one TFS collection at a time. This includes all of the projects in that collection.

### Depots

The tools are designed to import into a single classic Perforce depot. Any directory within that depot can be specified as the top level directory of the import structure.

## The Migration Process

The migration process is divided into three phases. Each phase is dependent on a successful completion of the previous phase. If you are migrating multiple collections the phases of each migration are independent.

As a review of the migration phases:

Extraction. Where metadata information is extracted from TFS.

Generation. Where extracted TFS metadata is used to generate an import control file.

Import. Where the generated import control file is used to coordinate the migration of information from TFS to Perforce.

### Extraction

During this phase the required TFS metadata is extracted from TFS.

The need for migrations to access the entire set of available TFS metadata makes for a time consuming process. Metadata extraction rates of 5K changesets per hour are typical.

To make the extraction process more efficient, specialized tools extract and associate relationship metadata as separate steps within this phase. These tools typically run at about 5K changesets per minute.

### Generation

During this phase the extracted TFS metadata information is used to generate an import control file.

Generation processing is fast. Typical performance is on the order of 5K changesets per minute.

Translating TFS metadata into import control statements is an exacting process that needs to account for a large number of side-effects, dependencies, and interactions. Because of this complexity modification of the generation code is strongly discouraged. Instead, create scripts to modify the generated import control file. Control file modification is typically used to accomplish tasks such as Transforming TFS user names into Perforce user names.

### Import

During this phase an import control file is used to transform TFS metadata and file content into Perforce metadata and file content.

Import is slow. Connectivity, the size of the files involved, the average number of files in a changeset, the average number of files in a branch, the number of files with more than 50 revisions, and other factors all impact import performance. A small TFS repository may import at close to 80 changesets per minute. For larger repositories typical import performance is often between 5 and 10 changesets per minute.

Import uses TFS and Perforce workspaces as a state mechanism. Workspaces require client disk space as described above. They also require special attention if an import needs to be restarted. Import provides a forced stop capability to exit with workspaces in a consistent state.

The ability to restart an import has several common uses:

Recover from import failure.

Mitigate user impact by only running the import during periods of low server load.

Manage schedules by completing an import using a final incremental stage.

See the *Special Procedures* section below.

# Process Details

This section describes the TFS full history migration process in detail.

## Installation

This section describes the most common methods for installing the tools and infrastructure software.

### TFS

Installation of client software is from standard Microsoft distributions.

Command line access to TFS is established using vsdevcmd.bat.

### Perforce

Installation of client software is from standard Perforce distributions.

See the *Best Practices* section for recommendations on default values.

### Perl

The most common Perl distribution for Windows is the community version of ActivePerl from ActiveState. Only the Perl scripting language portion of the install is required.

Allowing creation of an association with Perl and the .pl file extension is strongly recommended.

### Tools

See the *Best Practices* section for recommendations on install and access to the migration tools.

## Tool Characteristics

This section provides details to the more significant characteristics of tool operation.

### Output

Each phase generates volumes of output. This happens by default and usually has no mechanism to suppress. With long run times it is best to capture all available information in case it is needed. It can always be deleted after the migration has been completed and validated.

Most of the tools will not operate if an output file that they generate already exists. Tool options are provided to specify user defined file names when appropriate. This behavior is designed to protect against loss of information that may have taken a considerable amount of time to create. Wrapping the tools in scripts that automatically delete expected output files prior to operation is against best practice.

### File formats

All of the files generated by, or used as input to, the tools are text. This avoids special tools to decode binary formats. It also allows for examination and modification of generated files. There are tools within the distribution that perform common extraction, analysis, and adjustment processing for these files.

See CodeNotes.txt in the doc distribution directory for additional comments about the XML-like format of some content.

### Options and arguments

It is best to consider options and arguments as order and case sensitive.

All tools have these options:

–V to generate tool version information

–h to generate tool usage information

All tools contain perldoc.

Tools that output significant processing and progress status have a –l option to specify a log file that contains a duplicate of the generated output.

## Extraction

This section covers the extraction process. The extraction process uses multiple steps. The results of the various extraction steps are combined during the generation phase. Some of these steps extract information for one changeset at a time. Others extract information for multiple changesets concurrently. Using multiple extraction steps is at least an order of magnitude faster than extracting changeset information individually.

### Operational context

The extraction tools use the full access TFS account.

The tools supporting extraction operate from the top level directory of a workspace that references the desired TFS collection. The workspace should map $/ so as to reference all of the projects in a collection. The workspace does not need to be populated. The extraction tools will not alter the workspace population during extaction.

### Generation history

To extract the complete generation history for a repository you would use:

TFSHistory.pl -l log.h1 -r history.raw 1

The -l and -r options specify a log and extraction file respectively. The argument 1 specifies that the extraction should start with changeset 1. Since a second argument is not specified extraction continues until all known changesets are extracted.

To mitigate the impact of extractions on the TFS server you can perform extractions as multiple steps during times of low server utilization. For example, to extract a repository with 40,000 changesets you could perform the extraction as two steps with:

TFSHistory.pl -l log.h1 -r h1.raw 1 19999

TFSHistory.pl -l log.h2 -r h2.raw 20000

Note that two separate history extraction files - h1.raw and h2.raw - are used.

Extraction has been known to encounter access errors and frozen TFS user access. The tools exit with a status message if either of these events occurs. Resolve the failure and either re-run the entire extraction or re-start an extraction from the failing changeset.

Extract supports a coordinated stop. Simply create the file specified at the start of the output from TFSHistory.pl. The tool ignores any content within this file. When the tool detects the stop file it exits with a status message after completing extraction activities for the current changeset. To restart, delete the stop file and specify a restart from the changeset after the stop changeset.

### Single history file

There are many reasons that history content would not be extracted into a single file. However, the tools that use history extraction information expect all extracted history information to be present in a single file.

If multiple history files are extracted they need to be combined into a single history file. Changeset order is expected within the combined history extraction file. However, gaps in the changeset sequence are allowed to account for corrupt or unrecoverable changeset information.

The tool rawCombine.pl is provided to assist with combining history files while maintaining the detail information contained within that file. For example,

rawCombine.pl h1.raw h2.raw > history.raw

combines the two raw history files from the example above to create a single raw history file ‘history.raw’.

### Identify directories

There are situations where the generation process needs to differentiate between directories and files. Identify the directories in a collection with:

tf dir $/ /recursive /folders /deleted > tfdir.out

TFSDirFormat.pl tfdir.out > directories

‘directories’ is the default directory information filename used by TFSAssociate.pl.

If the request is too complex for the server to handle you can request directory information for the individual projects in a collection by replacing $/ with $/project-name. For example, if the collection contains two projects – p1 and p2 – you could use:

tf dir $/p1 /recursive /folders /deleted > tfdir.p1.out

TFSDirFormat.pl tfdir.p1.out > directories

tf dir $/p2 /recursive /folders /deleted > tfdir.p2.out

TFSDirFormat.pl tfdir.p2.out >> directories

The association tool expects a single directory information file so output from the second TFSDirFormat.pl command must append to the file ‘directories’.

### Identify merge sources and targets

Generation needs to relate merge sources to merge targets. Establish the merge information for the collection with:

tf merges $/ /recursive /noprompt /format:detailed > merges

This is likely to run for several minutes.

If the request is too complex for the server to handle you can request merge source and target information for the individual projects in a collection by replacing $/ with $/project-name. For example, if the collection contains two projects – p1 and p2 – you could use:

tf merges $/p1 /recursive /noprompt /format:detailed > merges

tf merges $/p2 /recursive /noprompt /format:detailed >> merges

The association tool expects a single merge information file so output from the second tf merges command appends to the file ‘merges’.

### Identify rename source and target

The generation tool needs to have rename sources related to rename targets. This requires a two-step sequence. In the first step renames are identified. In the second step the source and target information for identified renames are established. This example operates using default filenames:

TFSAssociate.pl -l log.a1 history.raw

TFSRenames.pl unresolved > renames

The amount of time required for TFSRenames.pl to run depends on the number of renames. However, it typically runs for several minutes. Renames are handled individually so there should be no possibility of requests being too complex for the TFS server.

### Identify two-hop rename source and target

When you rename a file more than once prior to check-in TFS usually tracks only the original and final names. However, there are scenarios where it tracks an intermediate name. Unfortunately these two-hop renames cannot be detected until generation. You identify that two-hop renames are present by error messages from the generation process.

Two-hop rename sources are related to their targets using a two-step sequence. In the first step the two-hop renames are identified. In the second step the source and target information for the two-hop renames are established. This example uses a log file (log.g1) created during the first generation pass to drive extraction of the two-hop rename information into the default two-hop filename ‘TwoHop’:

TFSTwoHop.pl log.g1 > TwoHop.names

TFSRenames.pl TwoHop.names > TwoHop

The amount of time required for TFSRenames.pl to run depends on the number of two-hop renames. Renames are handled individually so there should be no possibility of requests being too complex for the TFS server.

NOTE: Although the TFSRenames.pl tool is used the output needs to be captured in a file that is separate from normal rename information. Be sure that you don’t overwrite the rename extraction file or append two-hop information to that file.

## Generation

This section covers the generation process. Generation is completed with a single pass through the generation sequence unless two-hop renames are detected. If encountered, two-hop rename information needs to be extracted and the generation sequence repeated.

### Operational context

Generation is independent of TFS, Perforce, and workspace context.

### Generation

The initial generation pass is a two-step sequence. In the first step additional information is added to the raw extracted history information to create items with a more complete activity profile. In the second step the full profile items are used to generate a control file that can be used to coordinate a full history import. A typical first generation sequence is:

TFSAssociate.pl -l log.a1 -i items.1 history.raw

TFSGenerate.pl -l log.g1 items.1

If the output from TFSGenerate.pl includes messages that start with the text ‘+++ RenameMatching delete’ then there are two-hop renames. The example above produces the log file used in the two-hop extraction sequence above.

Extract the two-hop information as described above then perform this second pass generation sequence:

del unresolved

TFSAssociate.pl -l log.a2 -i items.2 history.raw

del cg.ctrl

TFSGenerate.pl -l log.g2 items.2

Note the delete commands in the above sequences. Delete of the file ‘unresolved’ removes a file generated by TFSAssociate.pl that identifies unresolved relationships. By default, TFSAssociate.pl appends information to this file. It is removed at this point to establish a baseline that can be used by incremental and other advanced procedures. Delete of the file cg.ctrl only occurs in the second sequence. This delete removes any previously generated control file.

## Import

Import is coordinated with the tool TFSImport.pl. It uses an import control file created during the generation phase.

### Operational TFS context

The import process uses the full access TFS account.

The import coordination tool uses a TFS workspace as a state mechanism. The workspace should map $/ to allow access to the entire collection. The import coordination tool will manage population of the workspace.

The environment variable TFS\_TLD establishes the path to the top level directory of the TFS workspace. If TFS\_TLD is not set then you must specify the –TFS option with every execution of the import coordination tool.

### Operational Perforce context

The import process uses the Perforce super user account. They must have a valid login that doesn’t expire for at least as long as the import is estimated to run. An unlimited timeout is a better choice.

The import coordination tool uses a Perforce workspace as a state mechanism. The tool should have exclusive control over the management of this workspace. Except when it is appropriate to one of the special procedures described below user interaction with this workspace is likely to corrupt an import in progress.

The import tool validates various characteristics of this workspace prior to operation. If any of the characteristics do no match requirements the import does not operate.

The workspace must specify these options:

Submitunchanged

Allow clobber

Support rmdir

The workspace must have a single line view mapping. The client side of the view mapping must be exactly ‘//client-name/…’. A typical import workspace view mapping for an organization that coordinates all imports into a common depot might look like:

//import/collection-A/… //import\_ws/…

### Tool context

There is a directory of execution for the import coordination tool. This directory should not be within the structures used by either the TFS or Perforce workspaces.

By default the import coordination tool creates a detailed progress message each time 10 changesets are migrated. These progress messages provide status about an import in progress. The environment variable TFS\_PROGRESSEVERY can be set to alter this default. You can also set the frequency using the –p option to the tool.

Import supports a coordinated stop. Simply create the forced stop file specified at the start of the output from TFSImport.pl. The tool ignores any content within this file. When the tool detects the stop file it exits with a status message after completing import activities for the current changeset. See the *Clean restart* topic in the *Special Procedures* section below for information about restarting the import.

### Initial import

This section walks through the sequence of steps that are recommended to initiate an import. Don’t give in to the temptation to skip the more obvious steps. Experience has shown that performing each of these steps is important to a successful import.

This example uses D:\import\collectionA as the directory of execution and import\_collectionA\_ws rooted at D:\ws as the Perforce workspace. Adjust as appropriate for your environment.

Open a command prompt.

Transition to the intended directory of execution and initialize operational context. Establishing operational context using a bat file is a best practice that ensures consistency.

Verify that the TFS workspace exists with:

cd /D %TFS\_TLD%

This is effectively what the import will do. The directory must exist. Be sure you are in the expected directory.

Verify that TFS commands work and that the context is appropriate to the import by displaying the working folder for the TFS workspace with:

tf workfold

Verify that the workspace is the expected workspace, the collection is the expected collection, and that the mapping is to ‘$/’.

IF THIS IS THE FIRST IMPORT clear the workspace and establish an initial state for the TFS workspace with:

tf get /v:1

DO THIS ONLY BEFORE THE FIRST IMPORT.

The TFS workspace directory structure should now be empty. Verify that it is empty with:

dir /s/b

The subdirectory $tf is there to support TFS. You need to figure out why any other “spare” files or directories exist then remove them. Extraneous files and directories may corrupt an import.

Now transition back to the import’s directory of execution:

cd /D D:\import\collectionA

Verify that Perforce commands work and that the context is appropriate to the import by displaying your Perforce context using:

p4 info

Verify that the user, workspace, and server are the expected values. If not, you need to resolve any differences.

Clear the workspace with:

p4 sync //import\_collectionA\_ws/…#0

Verify that the workspace has no open files with:

p4 -C import\_collectionA\_ws opened

Determine why there are any open files then revert them.

Transition to the root of the Perforce workspace and verify that it is empty with:

cd /D D:\ws

dir /s/b

There should be no files or subdirectories. You need to figure out why any “spare” files or directories exist then remove them. Extraneous files and directories may corrupt an import.

Finally, transition back to the intended directory of execution and initiate the import:

cd /D D:\import\collectionA

copy cg.ctrl cg.ctrl.1

TFSImport.pl -l log.i.1 cg.ctrl.1

This sequence uses the convention of naming control files sequentially and creating log file names that align with the control file name. This provides a baseline for the current import.

## Determining Success or Failure

The extraction and generation tools clearly indicate failure. Lack of failure indicates success.

When an import completes it generates a message indicating whether or not completion was due to an error. This is the primary method for determining if an import was successful. Regardless of success, all log files should be reviewed for indications that the tools have treated situations as warnings that you would consider an error.

Once started an import should run to completion. Messages generated during the import provide feedback about status and progress. Warning messages are not unusual. Warning messages indicate context that could be handled but was not expected by the import. Warning messages have many possible causes including side effects of the destroy command.

The import tool is designed to terminate any time it determines that the integrity of file content may have been jeopardized. The most common source of these failures is associated with communication problems. Note that this is distinctly different from situations, such as those created by destroy, where the required file content is available but the workspace context is inconsistent with the historic action being emulated. In these situations the import uses the file content and re-aligns workspace context after generating a warning message.

# Representation Considerations

There are features within TFS that either operate differently or have no direct Perforce equivalence. This section discusses how the import process deals with these situations. It is a best practice to address any representation issues prior to an import.

## File Types

Unlike Perforce, TFS file types do not specifically impact or define the storage or workspace content characteristics of files. Because of this the import process ignores TFS file types. Instead, import uses byte order mark (BOM) prefixes on file content to adjust the base type of imported files when appropriate. This is required to assure that future sync operations restore the appropriate content to workspaces. The attributes of a Perforce file type are never modified by the import process.

BOM prefixes on file content impact base Perforce types as follows:

|  |  |  |
| --- | --- | --- |
| ***Content BOM*** | ***Current Perforce base type*** | ***New Perforce base type*** |
| None | Text | --- |
| None | Binary | --- |
| None | utf16 | binary |
| UTF8 | text | --- |
| UTF8 | binary | --- |
| UTF8 | utf16 | text |
| LE16 | text | utf16 |
| LE16 | binary | --- |
| LE16 | utf16 | --- |
| BE16 | text | binary |
| BE16 | binary | --- |
| BE16 | utf16 | binary |

## Branch, merge and resolve

TFS branch, merge, and resolve are similar to their Perforce integrate and resolve counterparts. When the import requires an integrate or merge that yields the same results as Perforce the migration uses the appropriate integrate or resolve command.

Most instances of TFS branch and merge involving deleted files do not have a direct Perforce equivalent. When there is no equivalent, the TFS action is not replicated as part of the migration.

Likewise, TFS resolve is similar but not equivalent to Perforce resolve. When an import merge requires a resolve, import processing uses the MD5 signatures of appropriate TFS and Perforce workspace files to determine the resolve to emulate for the action. The possible combinations of MD5 signatures support these Perforce resolve actions:

Accept theirs (source)

Accept yours (target)

Accept merged (merge result)

Accept merge with edit

## Empty directories

TFS creates and manages empty directories. The migration does not attempt to emulate empty directories. Likewise, the migration does not emulate the rename or delete of empty directories.

## TFS destroy

There are two considerations relating to the TFS destroy command.

The first consideration is the ability of this command to generate one-sided branch, merge, and rename action history when the destroy removes only the source of an action. The migration generates the correct target file. However, the migration also generates warning messages relating to a missing source. These messages need to be reviewed in case they indicate a more serious problem.

The second consideration is use of this command between the time an initial extraction is taken and an extraction to support incremental import is taken. The impact of the TFS destroy command in this scenario is hard to predict. Almost certainly the TFS workspace will become inconsistent relative to the Perforce workspace. If possible, this command should not be used if an incremental import is planned.

## Character case

The character case of the workspace name must match the workspace specification even if the target Perforce server uses case insensitive handling.

If the target Perforce server uses case sensitive handling you need to make sure that the root specification for the import workspace exactly matches the character case on the client machine. This includes the drive letter – which must be upper case.

TFS support for case only renames impacts migrations. Case only renames are created with Perforce move commands when the target Perforce server is case sensitive. When the target Perforce server is case insensitive, case only renames are emulated with a delete and an add. The required deletes are performed as a separate “phantom” changeset emulation changelist. Regardless of how the server handles case the import tools maintain case consistency within the Perforce workspace.

# Special Procedures

This section describes procedures that may be required in addition to the standard import procedure described in detail above. Most of them require more than average skills using Perforce from the command line.

These procedures are classified as general and advanced. The advanced procedures may require expert level knowledge.

These descriptions are general by design. Each step in the description is required. If you don’t understand how to accomplish one or more of the steps described consider getting expert help.

## TFS and Perforce Tasks

This section provides an overview of the TFS and Perforce tasks required by one or more of the procedures in this section. It is intentional that the descriptions are short. Consider getting expert help if you don’t understand one or more of these descriptions.

You’ll need to identify any pending changelists associated with the import workspace.

You’ll need to list the description for a changelist.

You’ll need to identify any files in the import Perforce workspace that required add, edit or delete.

You’ll need to be able to revert files in a pending changelist.

You’ll need to be able to synchronize a TFS workspace to a specific changeset.

## General Procedures

Anyone capable of performing the tasks outlined above should be able to perform these procedures.

### Relating a changelist to an import sequence

The import process uses the TFS changeset number as a sequence reference. This changeset/ sequence value is specified in the last line of each import changelist description. The format of that line is always ==number== where number is the changeset/ sequence value.

Many of the procedures in this section require the ability to relate a changelist to an import sequence. To establish that relationship generate the description for the changelist and observe the changeset/ sequence value in the last line.

### User name management

Different organizations have different approaches to dealing with imported TFS user names. Some like to use the name as it existed in TFS. Some like to convert to a current Perforce user name.

Regardless of approach, user names must conform to appropriate Perforce requirements. The import tool enforces a default set of rules for user names and converts names that don’t conform into conforming names. This is done to avoid import failures.

The tool ctrlUserManager.pl is provided to assist with establishing user directed changes to TFS user names. In list mode it identifies all unique user names found within the control file. In mapping mode it translates names as specified.

See the tool perldoc for operational details.

### Path length management

Very long paths to files represent two potential problems. First, the path to the file in a workspace must be supported by the Windows client. Since the file exists in TFS you know that a workspace can be specified that will support it. However supporting and being usable are two different issues. Second, if you’re running a Windows Perforce server the depot storage path for the file must be supported.

The tool ctrlPathLength.pl is provided to assist with the detection and mitigation of path names that are too long for your environment. In list mode it identifies all unique paths that are 240 characters or longer. The 240 character threshold is a default. Other values can be specified. In skip mode, any import action that references specified file paths are changed to LENGTH actions. LENGTH actions create import log entries but are otherwise unprocessed.

In general, skipping actions during the import is not desirable. However, it is the only practical solution to the length problem. If the files aren’t important then skip may be the correct solution. If the files are important, they can be manually added after the primary import has completed. Regardless of how the file is added, issues associated with long paths persist. Skip leaves a gap in the history. Relocation has relationship history issues. Renaming has identification issues. Creating a long path creates a workspace root dependency for future users. The approach you take will be specific to your environment. You may even deal with different files in different ways. What can’t be allowed is an unrecoverable import failure caused by a long path.

See the tool perldoc for operational details.

### Clean restart

As detailed above, you can force an import to stop after it has completed processing migration of the current changeset. The actual time the import takes to stop depends on the amount of overhead associated with migration of the current changeset. After a successful forced stop a clean restart should be straightforward. Regardless, there are checks that should be performed.

To perform a clean restart after a forced stop:

Verify that the stop was clean. A stop was clean if the import workspace has no open files and no pending changelists. If the stop was not clean perform a failure restart.

Determine the last imported TFS changeset from the description of the last submitted changelist associated with the import workspace.

Transition to the top level directory of the TFS workspace. Verify that the TFS workspace is consistent with the latest import by specifying a TFS get of the last migrated changeset. That get should report no changes. If changes are reported perform a failure restart.

Verify that there are no files in the Perforce workspace that need to be added, edited, or deleted. If there are any such files perform a failure restart.

Transition back to the import coordination directory.

Delete the forced stop file.

Create a restart control file with ctrlList.pl using the most recent control file as a base. The first sequence is the next sequence after the last imported TFS changeset. The last sequence is an arbitrarily large number greater than the most likely last changeset reference (999999 is usually a good choice). For example, if the most recent control file was called cg.ctrl.3 and the last imported changeset was 2345 then generate the restart control file with:

ctrlList.pl cg.ctrl.3 2346 999999 > cg.ctrl.4

This is consistent with the best practice of sequential control file naming.

Restart the import. For the example above, the best practice form of the command would be:

TFSImport.pl -l log.i4 cg.ctrl.4

### Incremental import

Incremental import is actually a variation on the clean restart procedure described above. The primary difference is the addition of additional extraction and generation steps.

To perform an incremental import:

Verify that the most recent import ended cleanly. An import has ended cleanly if the import workspace has no open files and no pending changelists. If the most recent import did not end cleanly an incremental import is still possible. However it is beyond the scope of this document to cover all of the possible issues that need to be addressed to achieve it.

Save a copy of all working directory files created to this point. This provides a set of sources that provide recovery in case mistakes are made in the process that follows.

Determine the last imported TFS changeset from the description of the last submitted changelist associated with the import workspace.

Extract history starting with the next changeset in sequence. For example, if the last imported changeset was 4567 then the history extract would like look:

TFSHistory -l log.hi -r hi.raw 4568

After the extract append the new raw history information in ‘hi.raw’ to the baseline raw history information in ‘history.raw’.

Re-extract the directory information.

Re-extract the merge source and target information.

Clear any unresolved association information by deleting the file unresolved. The rename processing that follows depends on new unresolved association information to avoid duplicating rename and two-hop entries.

Extract rename source and target rename information associated with the new history. The command sequence would look like:

TFSAssociate.pl -l log.ai history.raw

TFSRenames.pl unresolved >> renames

history.raw should be the previous history.raw file with the incremental history appended to it.

NOTE the >> which appends to renames

Like the rename information, adapt the two-hop sequence to append to the file TwoHop

Generate the control file.

Now follow the clean restart sequence described above.

## Advanced Procedures

These procedures are more challenging than the general procedures. The challenge is not the procedural steps. Rather, the challenge is identifying if problems being addressed are benign or indicative of a more serious and potentially corrupting problem. Because of the long elapsed times that imports typically require, corrupting a migration needs to be avoided at all costs. In particular, import into a production environment needs additional scrutiny because of the difficulties associated with reverting the large number of changes associated with an import.

### Failure restart

Determining if you should attempt to restart after a failure depends on the nature of the failure. If messages from the import tool indicate communication failure or a freeze of the TFS interface then a failure restart can be attempted once communication is reestablished. Other failures should be addressed with the appropriate support forum as they are indicative of potential corruptions or tool failure.

Overall, the failure restart procedure primarily depends on whether or not the most recent changelist for the import workspace is pending. However, if your Perforce server uses case insensitive handling the failure may have occurred during the second phase of a case only rename emulation. To determine if this is the situation, examine the description for the last successful submit. If the description ends with:

==NNN==

case only rename setup

rather than the usual

==NNN==

then the failure was during the second phase of a case only rename emulation. To recover from a case only rename emulation follow this procedure at the appropriate point in the recovery procedures below:

**NOTE: Recovery of failures during the second phase of a case only rename sequence involves obliterate and other potentially risky activities. Only experienced Perforce users should attempt this procedure.**

The last successful submit is the rename emulation changelist.

Sync the changelist files to #0.

Save the list of files in the changelist. This list will not be available after the obliterate in the next step.

Obliterate the file versions for the rename emulation changelist. ONLY obliterate the file versions in this one changelist. Obliterating other versions corrupts the migration.

Delete the emulation changelist.

Sync the Perforce workspace to the enumlation changelist.

Remove the last line of the changeset to changelist mapping file (cs2cl.map by default). The changeset number in this line should have a C appended to the changeset number. If it doesn’t you’ve probably made a mistake.

The #0 then #head sequence is important to restoring appropriate legacy character case in the workspace. If the file list is very large then it is likely that this is a case only rename of a directory. You can save yourself a lot of effort if you use ‘…’ in your Perforce specifications. Look at the sequence (using ctrlList.pl). If the attribute action of the AI tags prior to the PROCESS tag is renamecase-P then this is a directory rename. The reference attribute identifies the name of the directory after the rename.

If the most recent changelist is pending follow this procedure:

Determine the TFS changeset in progress from the description of the pending changelist associated with the import workspace.

Revert the files in the pending changelist.

Delete the now empty pending changelist.

If needed, perform the rename emulation procedure described above.

Identify any files in the Perforce workspace that need to be edited or deleted. If there are any such files the Perforce workspace is inconsistent relative to the import. The revert should have eliminated any such files so they are indicative of potential corruption. If you decide to continue you should recover workspace consistency by forcing the workspace to sync to the last successfully submitted changelist.

Identify any files in the Perforce workspace that need to be added. Files that need to be added are not necessarily indicative of corruption because revert does not modify files opened for add. Delete these files from the workspace.

Synchronize the TFS workspace to the changeset prior to the one being processed when the failure occurred. It is likely that changes will be reported but this it isn’t a requirement.

Create a control file that starts with the changeset in progress as identified at the start of this procedure.

Restart the import with the control file just created.

If the most recent changelist was a successful submit follow this procedure:

Determine the last imported TFS changeset from the description of the last submitted changelist associated with the import workspace.

If needed, perform the rename emulation procedure described above.

Identify any files in the Perforce workspace that need to be edited or deleted. If there are any such files the Perforce workspace is inconsistent relative to the import. This is indicative of corruption because all such files should be associated with a pending changelist. If you decide to continue you should recover workspace consistency by forcing the workspace to sync to the last successfully submitted changelist.

Identify any files in the Perforce workspace that need to be added. Files that need to be added are not necessarily indicative of corruption because processing may have created them but not yet requested the add. Delete these files from the workspace.

Synchronize the TFS workspace to the last imported TFS changeset. It is likely that changes will be reported but this isn’t a requirement.

Create a control file that starts with the changeset following the changeset identified at the start of this procedure.

Restart the import with the control file just created.

### Selected projects

Import of selected projects is not directly supported by the tools. The sources to the tools are available. This topic provides an overview of the challenges associated with providing support for such a capability should you decide to modify them.

History extraction will remain effectively unchanged. A filter for the extracted history information would be required to reduce the processing set to only those changesets that involve files and directories in desired projects. A filtered history extraction would implicitly filter rename and two-hop extractions. The directory and merge extractions would need to be done relative to the projects of interest.

Generation should be largely unchanged. It will generate based on the extraction information it is provided. However, there are likely to be issues with changeset actions where either the source or target but not both reference a file in a desired project. Unlike similar destroy scenarios both the source and target create extraction information since they both still exist.

Workspace integrity validation is an integral part of import processing. The current integrity validation processing is not designed to deal with workspaces containing only a subset of a collection. A dependency would be created between the import and TFS workspace mappings.

# Best Practices

This section outlines some of the best practices associated with TFS migrations. These practices have been developed from experience with TFS and other migrations.

## Import Users

Other than needing unrestricted read access to the entire set of files within migration collections there is nothing special about the TFS import account. Except for creating import workspace(s) they will never update the TFS server.

A Perforce user that is only used to support the migration typically makes management tasks easier. A name such as ‘import’ is suggested. The user name will only be associated with pending changelists. It is replaced by the TFS user name after the changelist is submitted.

Create a group specifically for this import user. Do not include the user in other groups. Use the group within the protections table. After import has been completed the import user, their group, and their protections table entries can be removed.

A single user can support concurrent import of multiple collections.

## Import Workspaces

Create unique workspaces for each collection being imported. Use a distinctive name such as import\_COLLECTIONNAME\_ws. The Perforce workspace name will persist as the workspace associated with a changelist. The distinctive name helps users identify file versions that originate from an import.

Windows paths have a 260 character limit. It is not unusual to find long (240+ character) paths in the history of a TFS repository. The directory used by the TFS workspace and the directory specified as the Perforce workspace root often need to be short for example - c:\TFS and c:\P4. Making these root directories short and unique when importing multiple collections needs to be considered.

## Directory Structures

Create a unique top level directory to contain all non-workspace related files. Under that directory install the files from the distribution source (currently LICENSE.txt and the directories src and doc). Create a unique directory for each collection being migrated as a peer to the src and doc distribution directories. All extraction, generation, and import files associated with the import of a specific collection should be contained within the directory designated for import of that collection. Don’t attempt to mix multiple collections in a single directory by using unique names.

## Operational Context

Use the same techniques for establishing operational context for all types of import. The techniques can be refined during POC or test imports but should be stable by time a production import is attempted.

Establish a bogus username, server port, and workspace name as the default Perforce user values.

Likewise, don’t include TFS executables, Perforce executables, or tools in the default PATH.

All of these defaults are intended to force a controlled context initialization as part of the import process.

Be familiar with the output from these commands and use them to verify operational Perforce context prior to every import:

p4 set

p4 info

## Import Depots

Some organizations use a unique depot for each collection being imported. Others use collection relative branches within a common import depot. Your choice should align with your organizational standards. The tools don’t care.

If you have a Windows based Perforce server and you have long path names in collections keep in mind the depot storage overhead added to those paths by the server.

Make the imported information read-only after the import has completed. Create new branches or streams from the imported files for post-import work. Having a clean split between import activity and post-import activity will prove useful for development and support activities.

## Automation

There is a temptation to automate processes such as restart. Avoid this temptation. Rather, create scripts that support your style of manual process. Some rules of thumb:

Use a script to establish context for a command window. A script that is run once after a command window is first initialized is often best.

Create unique versions of intermediate files. Only “clean” these intermediate files after your import has completed. The resulting audit trail of intermediate information can be critical to resolving issues.

When command output is redirected to a file have the script verify that such a file does not exist to avoid overwrite of critical data.

Avoid scripts that delete generated files as part of a processing sequence. Rather, create explicit “reset” scripts to delete any such files.

Consider creating a script to save the current state of all files generated or consumed by extract, generation, and import. These files are critical to restart and incremental imports.

You’ll note that the recommended automation approach is to use disk space rather than risk the loss of information. Much of the information takes significant time to create. Disk space is usually easy to find. Schedule time lost re-creating information is very hard to recover.

## Performance

Spending a lot of time optimizing import performance is generally counter-productive. Regardless, certain best-practices can significantly improve the chance that your import will operate at or near maximum performance levels. If resources are a constraint, then consider applying these best practices only to the production import where elapsed time and scheduling tend to be more critical.

Run the extraction from a client with a local network connection to the TFS server.

Generation tends to use significant amounts of memory. It is best run on 64-bit clients using a 64-bit perl distribution. 32-bit clients and 32-bit perl distributions have been known to fail due to memory exhaustion.

Run the import from a client that has a local network connection to both the TFS server and the Perforce server. If that’s not possible, have the remote connection to the TFS server. Because of their unique operational characteristics it is best if you do not consider Perforce access through proxies or replicas to be the same as local access.

Imports are automation that is performing atypical operational sequences with a high frequency. Although only one user account is used by the various import phases, imports typically generate server loads that are equivalent to the load imposed by a large number of active human users. Imports also require significant amounts of memory and they tend to access every file in a disk structure multiple times. It is unlikely that your systems are optimized to support this type of activity. Keep these factors in mind if you are using VMs. If you’re not using VMs neither the TFS server nor the Perforce server make good choices for hosting the clients involved in the migration.

## Import Everything

Any method for creating a subset of the import information creates a potential for error. Although importing everything takes more time, the net schedule impact should be minimal because of the incremental import capabilities.

The import everything approach creates a clear audit relationship between the information in Perforce and the information in TFS. It also eliminates the complexities associated with importing a previously “unwanted” project after a primary import has already been completed.

# Tool-kit Tools

This section is a brief summary of the tools provided with the tool-kit. Many of these tools have been used in context above.

## Content Evaluation and Management

These tools are commonly used to support the special procedures described above:

ctrlList.pl

ctrlPathLength.pl

ctrlUserManager.pl

These tools are most useful in support and diagnostic scenarios:

FileDetails.pl – generate BOM and MD5 signature information for a file using import tool techniques.

rawList.pl – list changeset specific information in extract and generation files.

TFSActions.pl – generate TFS action profiles.

The perldoc for these tools contain examples and operational details.

## Extraction

These tools are required by the extraction phase:

rawCombine.pl

TFSHistory.pl

TFSDirFormat.pl

TFSRenames.pl

TFSTwoHop.pl

## Generation

These tools are required by the generation phase:

TFSAssociate.pl

TFSGenerate.pl

## Import

These tools are required by the import phase:

TFSImport.pl

## Comparing Migration Results

These are basic tools that provide a high-level comparison of import results:

profile.pl

profileCompare.pl

The perldoc for these tools contain examples and operational details.

# Notices

Originally developed for Perforce by VIZIM (www.vizim.com).

Copyright (c) 2014 Perforce Software, Inc. and VIZIM Worldwide, Inc. All rights reserved.

Please see LICENSE.txt in top-level folder of this distribution for license information